		Physical Format 105		Logic Format 106			
	block allocation	block number 107	relative block number 108	relative logic block number 109	logic block number 110	relative logic sector number 111	logic sector number 112
window-based region 120	window#0 100	0~511	0~511	0~511	0~511	0~16383	0~16383
	window#1	512~1023	0~511	0~511	512~1023	0~16383	16383~32767
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	window#14	7168~7679	0~511	0~511	7168~7679	0~16383	229376~245759
	window#15	7680~7999	0~320	0~320	7680~7999	0~10271	245760~255999
redundant reserved region 121	dynamic-link area 101	8000~8191	512~703				
	window-information area 102						
	dynamic-link information 103						
	boot-information area 104						

FIG. 1

Item Name		Data Length (in bytes)
relative logic block number	**201**	2
window number	**202**	2
cycle counter	**203**	1
phase-lock flag	**204**	1
check sum code	**205**	1
data error flag	**206**	1
block error flag	**207**	1
error correction code	**208**	6

FIG. 2

Item Name		Data Length (in bytes)
relative block number	301	2
relative logic sector number	302	2
window number	303	2
writing block cycle counter	304	1
check sum code	305	1
window information cycle counter	306	1
block error flag	307	1
error correction code	308	6

FIG. 3

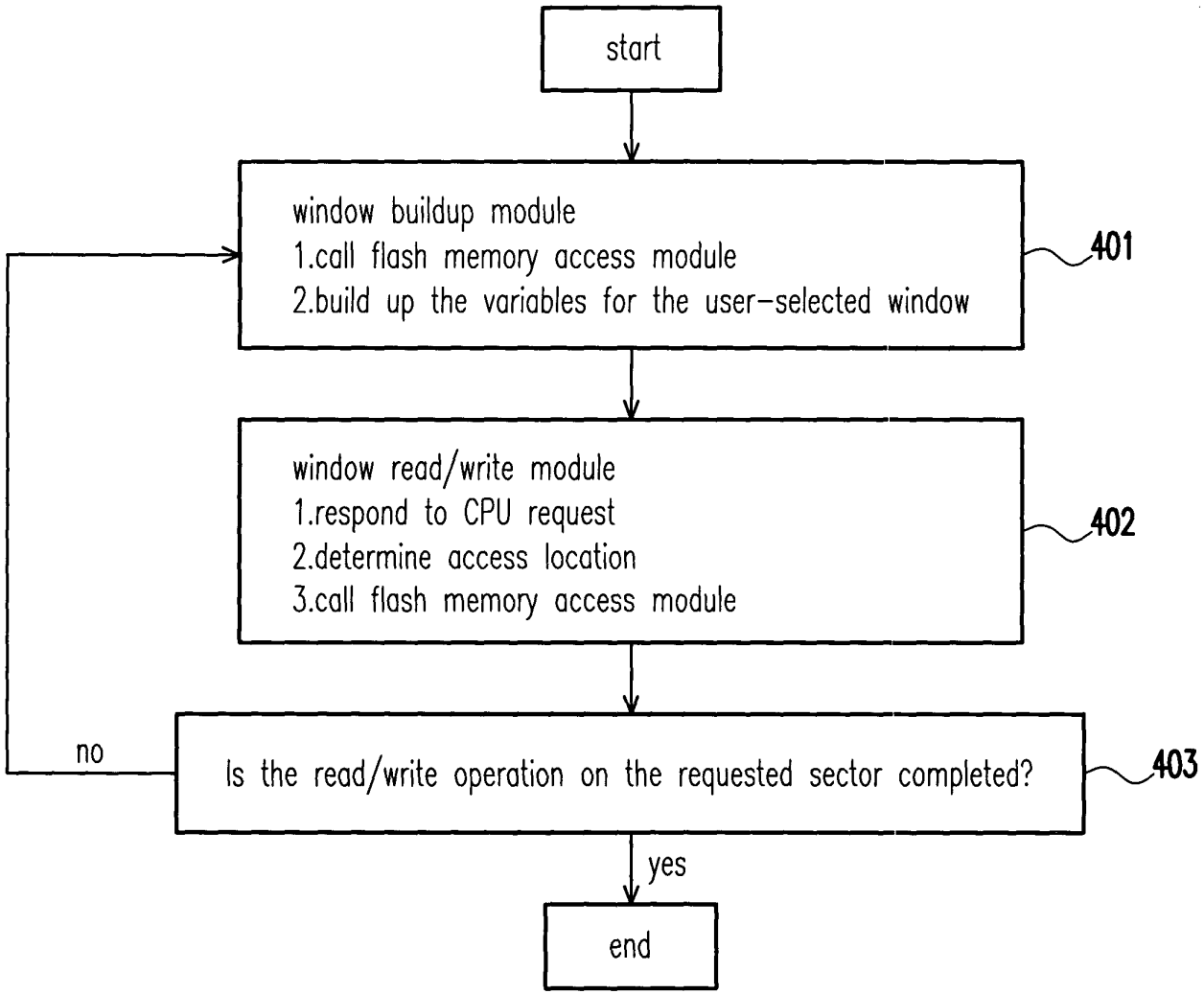
start
window buildup module
1.call flash memory access module
2.build up the variables for the user-selected window
401
window read/write module
1.respond to CPU request
2.determine access location
3.call flash memory access module
402
Is the read/write operation on the requested sector completed?
403
no
yes
end

FIG. 4

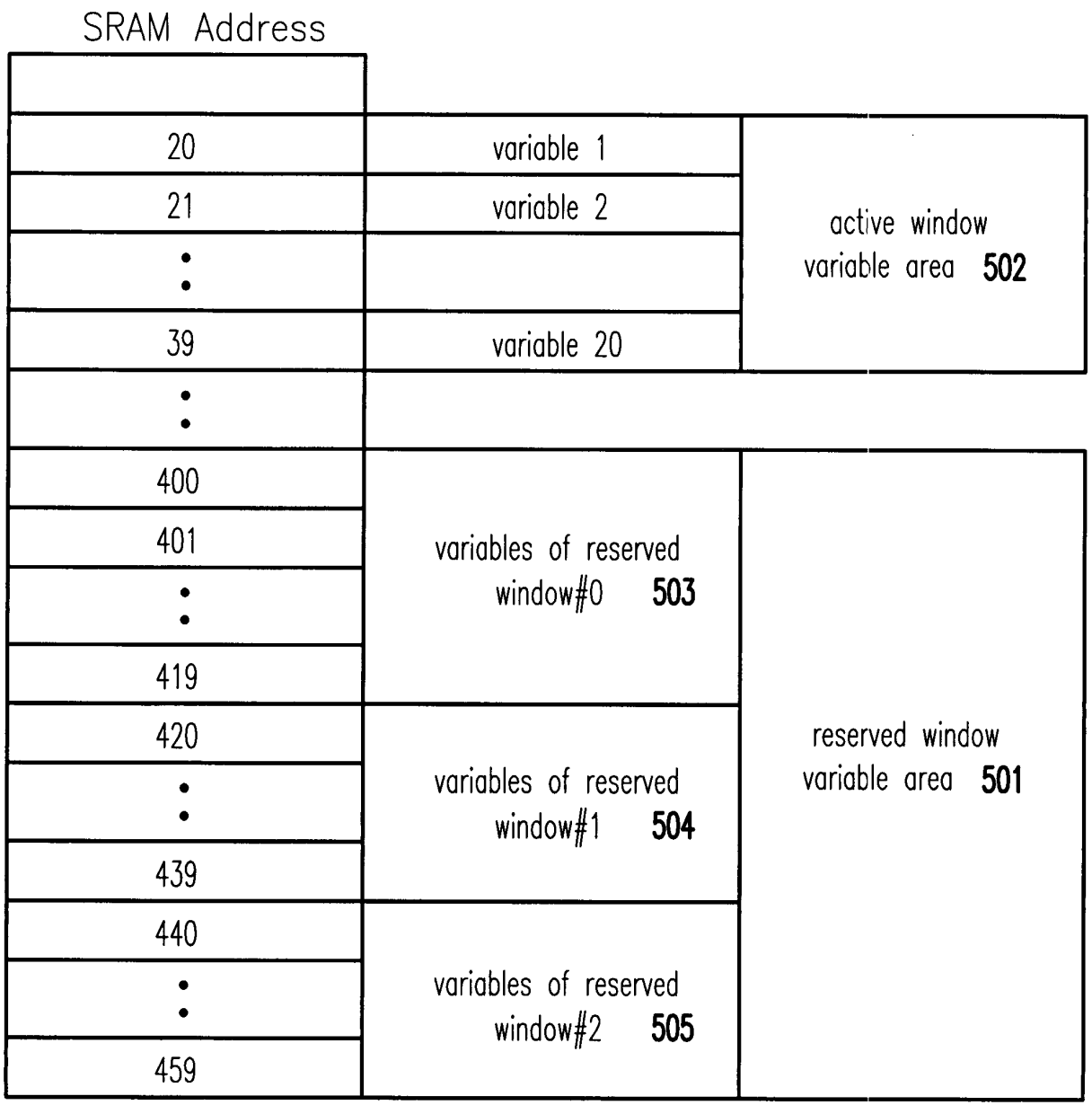
SRAM Address
20
21
39
400
401
419
420
439
440
459
variable 1
variable 2
variable 20
active window variable area 502
variables of reserved window#0 503
variables of reserved window#1 504
variables of reserved window#2 505
reserved window variable area 501

FIG. 5

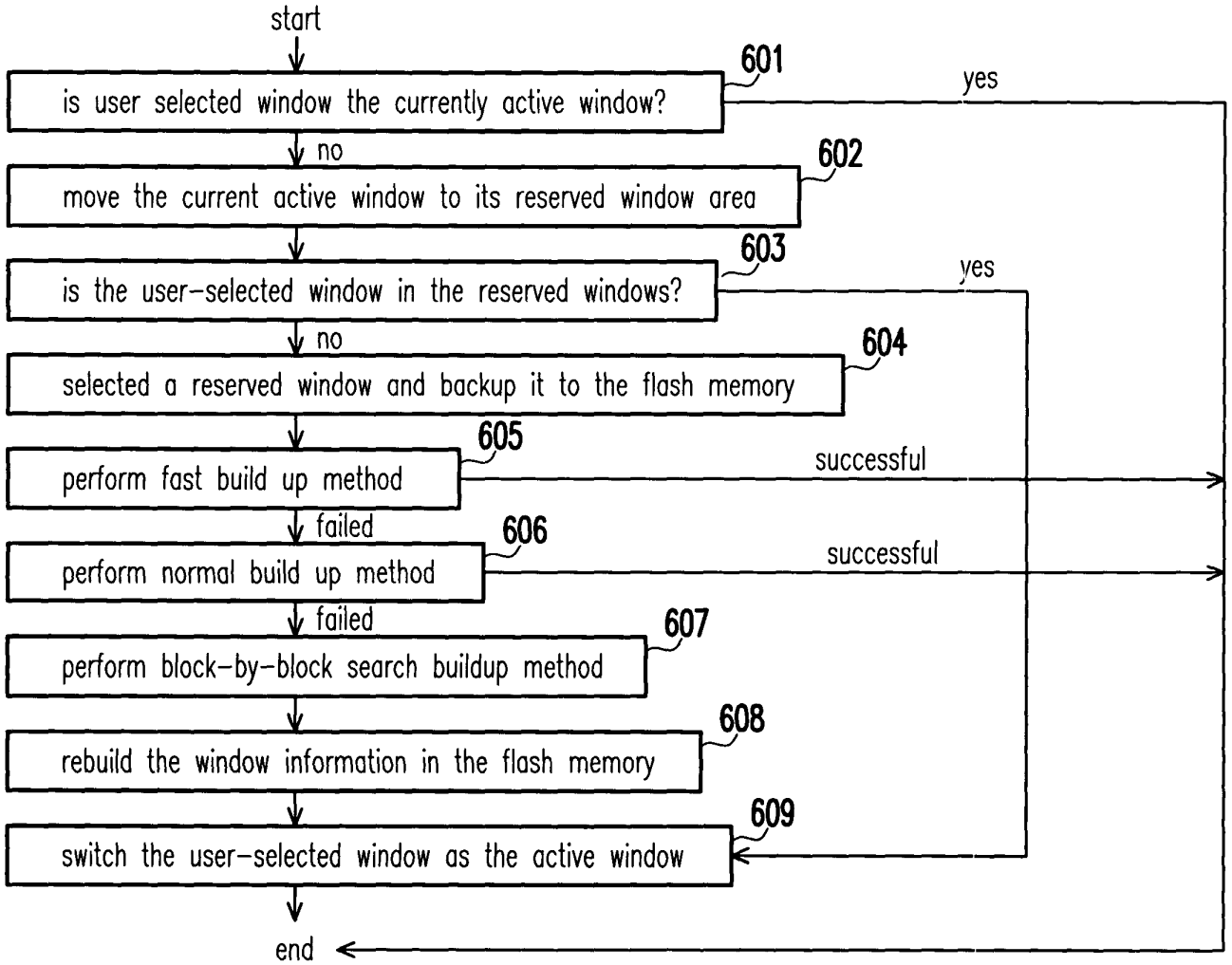
start
601
is user selected window the currently active window?
yes
no
602
move the current active window to its reserved window area
603
is the user-selected window in the reserved windows?
yes
no
604
selected a reserved window and backup it to the flash memory
605
perform fast build up method
successful
failed
606
perform normal build up method
successful
failed
607
perform block-by-block search buildup method
608
rebuild the window information in the flash memory
609
switch the user-selected window as the active window
end

FIG. 6

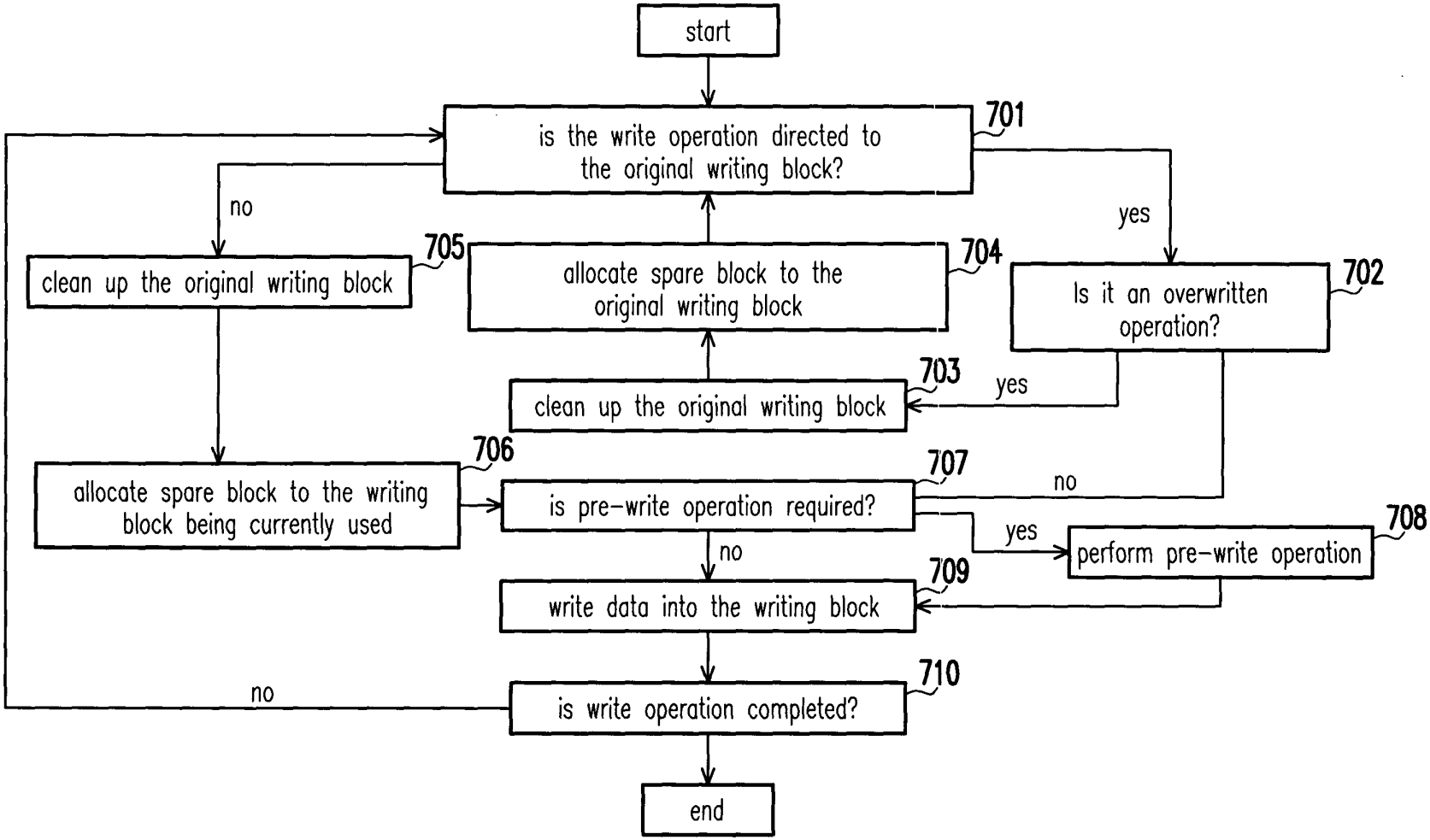
start
is the write operation directed to the original writing block?
701
no
yes
clean up the original writing block
705
allocate spare block to the original writing block
704
Is it an overwritten operation?
702
clean up the original writing block
703
yes
allocate spare block to the writing block being currently used
706
is pre-write operation required?
707
no
yes
perform pre-write operation
708
no
write data into the writing block
709
is write operation completed?
710
no
end

FIG. 7

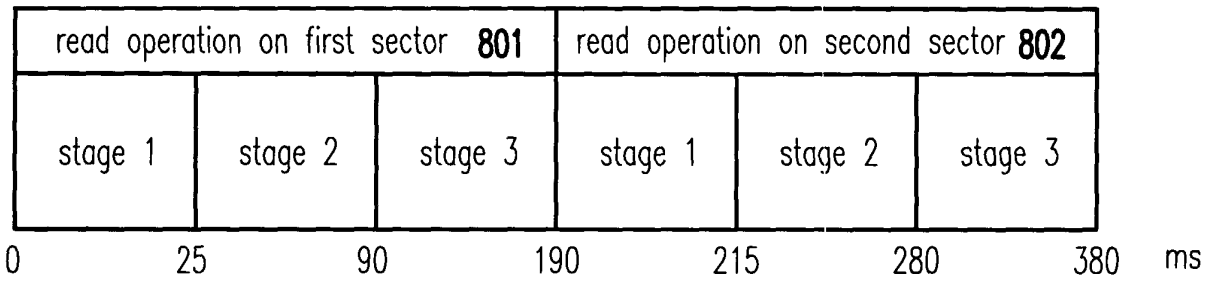

FIG. 8 (PRIOR ART)

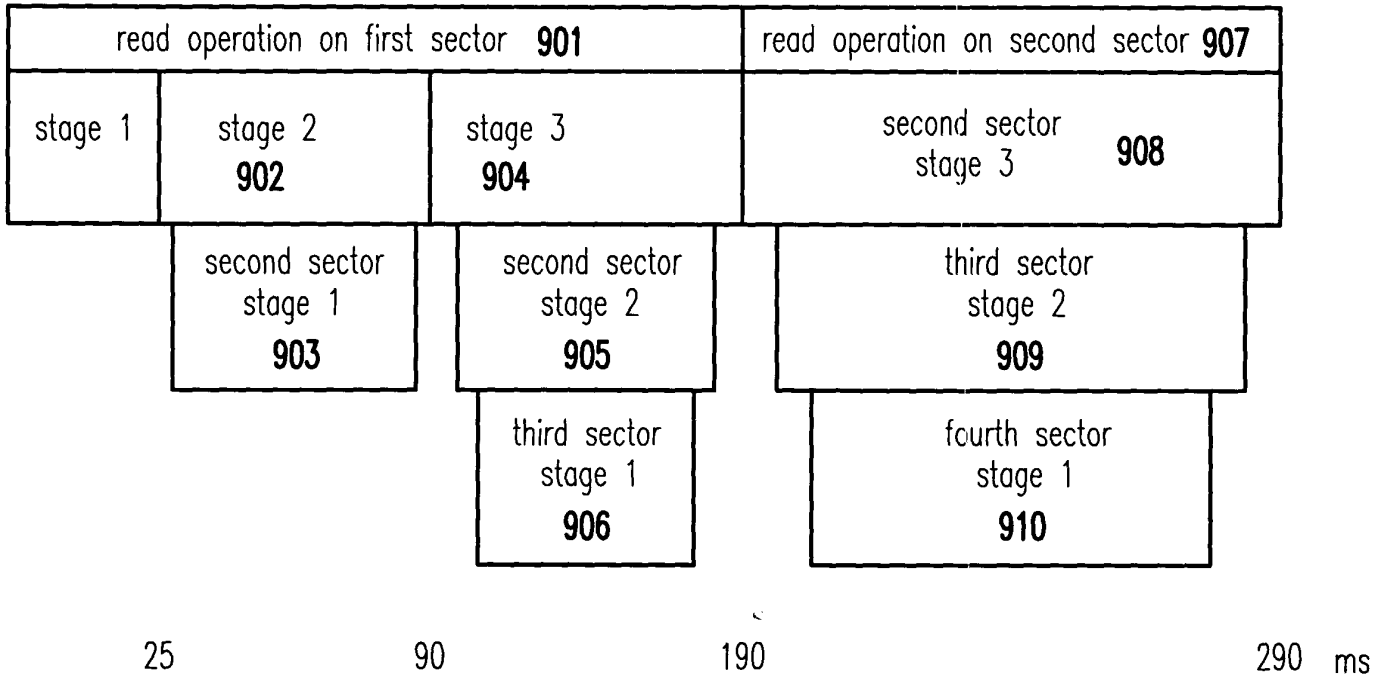
read operation on first sector 901
read operation on second sector 907
stage 1
stage 2
902
stage 3
904
second sector
stage 3
908
second sector
stage 1
903
second sector
stage 2
905
third sector
stage 2
909
third sector
stage 1
906
fourth sector
stage 1
910
25
90
190
290
ms

FIG. 9

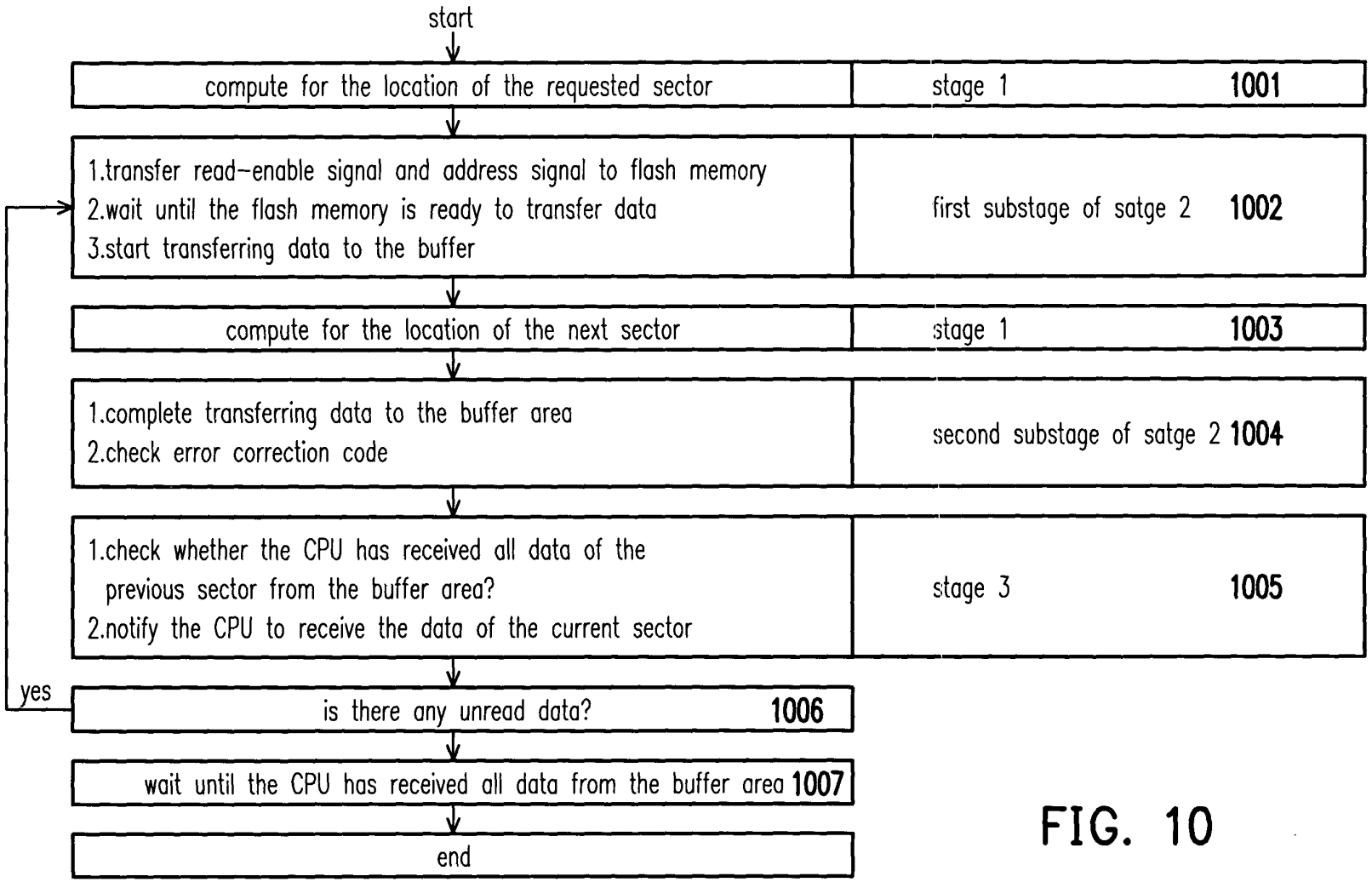
start
compute for the location of the requested sector
stage 1
1001
1.transfer read-enable signal and address signal to flash memory
2.wait until the flash memory is ready to transfer data
3.start transferring data to the buffer
first substage of satge 2
1002
compute for the location of the next sector
stage 1
1003
1.complete transferring data to the buffer area
2.check error correction code
second substage of satge 2
1004
1.check whether the CPU has received all data of the previous sector from the buffer area?
2.notify the CPU to receive the data of the current sector
stage 3
1005
yes
is there any unread data?
1006
wait until the CPU has received all data from the buffer area
1007
end

FIG. 10

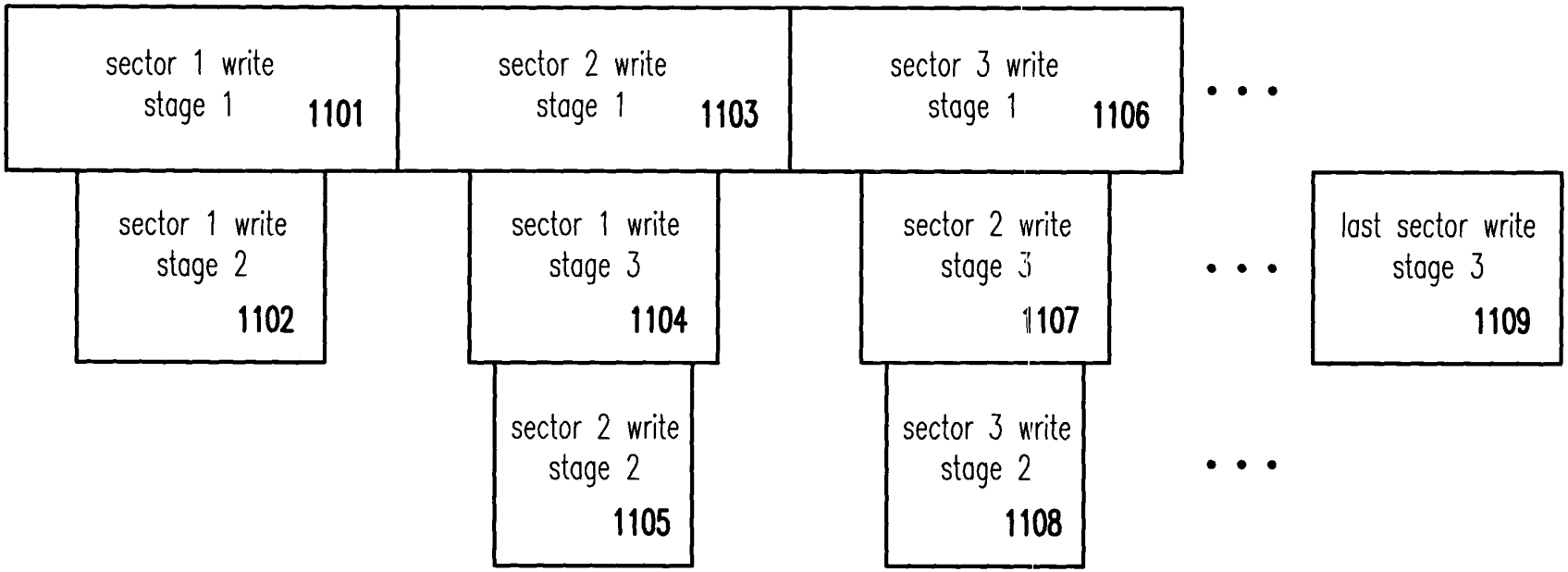
sector 1 write stage 1 1101
sector 2 write stage 1 1103
sector 3 write stage 1 1106
sector 1 write stage 2 1102
sector 1 write stage 3 1104
sector 2 write stage 3 1107
last sector write stage 3 1109
sector 2 write stage 2 1105
sector 3 write stage 2 1108

FIG. 11

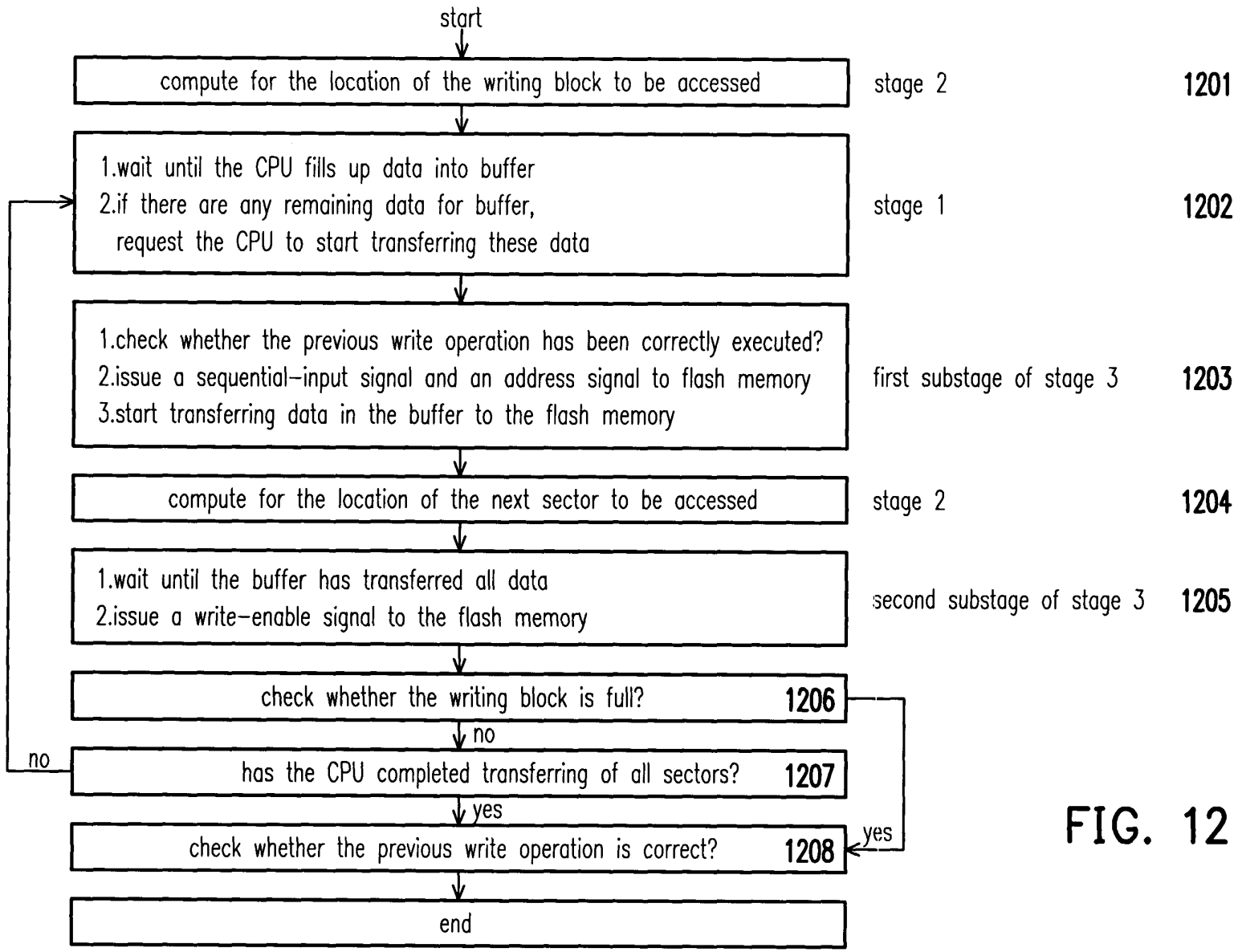
start
compute for the location of the writing block to be accessed
stage 2
1201
1.wait until the CPU fills up data into buffer
2.if there are any remaining data for buffer,
request the CPU to start transferring these data
stage 1
1202
1.check whether the previous write operation has been correctly executed?
2.issue a sequential-input signal and an address signal to flash memory
3.start transferring data in the buffer to the flash memory
first substage of stage 3
1203
compute for the location of the next sector to be accessed
stage 2
1204
1.wait until the buffer has transferred all data
2.issue a write-enable signal to the flash memory
second substage of stage 3
1205
check whether the writing block is full?
1206
no
yes
has the CPU completed transferring of all sectors?
1207
no
yes
check whether the previous write operation is correct?
1208
end

FIG. 12